ISBN 978-83-60810-22-4

A Branch-and-Cut-and-Price Algorithm for a Fingerprint-Template Compression Application

Andreas M. Chwatal*, Corinna Thöni*, Karin Oberlechner*, Günther R. Raidl*
*Vienna University of Technology, Institute of Computer Graphics and Algorithms
1040 Vienna, Austria
Email: chwatal@ads.tuwien.ac.at, corinnathoeni@gmail.com,
karin.oberlechner@gmail.com, raidl@ads.tuwien.ac.at

***Abstract*—In this work we present a branch-and-cut-and-price algorithm for the compression of fingerprint minutiae templates, in order to embed them into passport images by watermarking techniques as an additional security feature. For this purpose the minutiae data, which is a set of characteristic points of the fingerprint, is encoded by a spanning tree whose edges are encoded efficiently by a reference to an element in a dictionary (*template arc*) and a small correction vector. Our proposed branch-and-cut-and-price algorithm creates meaningful template arcs from a huge set of possible ones on demand in the pricing-procedure. Cutting-planes are separated in order to obtain connected subgraphs from which spanning trees can then be easily deduced. Our computational experiments confirm the superior performance of the algorithm in comparison to previous approaches for the spanning tree based encoding scheme.**

I. Introduction

THIS work is based on [1], where the authors introduced a combinatorial model to perform data compression specifically for a (small) set of unordered, multidimensional data-points. The need for such a kind of compression arises for instance when fingerprint minutiae data should be encoded in a way permitting this data to be embedded into images by watermarking techniques as additional security feature. For a detailed description of the application background we refer the reader to [1].

The compression model is based on a special kind of tree structure, connecting a subset of certain size of the given nodes. The nodes themselves correspond to the set of input data-points, i.e. the fingerprint minutiae. Compression is achieved by computing a suitable small set of *template arcs*, which enable a more efficient encoding of the intended tree structure. The compression model can thus be seen as a dictionary approach. Decompression is achieved by reconstructing the according subset of data-points from this particular tree structure. In section II we review the problem model.

In this work we focus on a more efficient solution of the underlying combinatorial optimization problem (COP). The model is actually a combination of two well-known COPs, the *minimum label spanning tree problem* [2] and the *k-cardinality minimum spanning tree problem* [3]. The resulting problem is called *k-minimum label spanning arborescence* (k-MLSA) problem.

The contribution of this article is a *branch-and-cut-and-price* (BCP) framework, detailed in section III, to solve the k-MLSA problem, and has for the first time been investigated in the theses [4]–[6]. Within this framework we use cutting-planes to obtain validity of the initially incomplete model and dynamically create and add new label variables in the pricing phase. In section III-A we present efficient methods to solve the according pricing subproblem, and the experimental results in section IV show that the presented exact algorithm is able to solve the problem to provable optimality within a reasonable amount of time for practical purposes.

II. Problem Model

The following general compression model as well as the underlying combinatorial optimization model have originally been proposed in [1]. The input data is given as a vector of n d-dimensional points $V = \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}\}$ from a discrete domain $\mathbb{D} = \{0, \ldots, \tilde{v}^1 - 1\} \times \ldots \times \{0, \ldots, \tilde{v}^d - 1\}$, with $\mathbb{D} \subseteq \mathbb{N}^d$. In our application these points correspond to the given minutiae data, where a single minutia is defined as a 4-tuple (x, y, θ, t) with x and y beeing the Cartesian coordinates of the point, θ its orientation and t its type. Within our compression model we usually only use x and y and eventually θ, and store the further data unprocessed.

Our aim is to select a subset of exactly k points to be stored in the compressed fingerprint template; i.e., our compression is not lossless, but with a suitable selection of k this can usually be considered sufficient for a reliable verification, i.e. matching to the features extracted by a fingerprint scanner. For this purpose we start with a complete directed graph $G = (V, A)$ with $A = \{(u, v) \mid u, v \in V, u \neq v\}$ on which we search for an optimal directed tree (outgoing arborescence), spanning at least k nodes, by optimization. Thus, all nodes and arcs in the graph have a topological as well as a geometrical interpretation. In the following, we emphasize the context we primarily refer to by denoting these elements in normal and bold letters, respectively.

Furthermore, we use a small set T of template arcs which act as dictionary elements for our compression approach. Instead of directly storing the coorinate values of the mutual difference vectors of the tree nodes for each tree arc, we encode these arcs by a reference to a *template arc* in combination with a *correction vector* from a prespecified small domain. Compression is achieved by optimizing the selection of the k points, as well as building up a feasible dictionary of template arcs T of minimal cardinality.

Consequently, a solution to our problem consists of

978-83-60810-22-4/$25.00 © 2011 IEEE

1) an ordered set of template arcs $T = (\boldsymbol{t_1}, \dots, \boldsymbol{t_m}) \in \mathbb{D}^m$, and
2) an outgoing arborescence $G_T = (V_T, A_T)$ with $V_T \subseteq V$ and $A_T \subseteq A$ connecting exactly $|V_T| = k$ nodes, in which each tree arc $(i,j) \in A_T$ has associated
   - a template arc index $\kappa_{i,j} \in \{1, \dots, m\}$ and
   - a correction vector $\boldsymbol{\delta_{i,j}} \in \mathbb{D}'$ from a prespecified, small domain $\mathbb{D}' \subseteq \mathbb{D}$ with $\mathbb{D}' = \{0, \dots, \tilde{\delta}^1 - 1\} \times \dots \times \{0, \dots, \tilde{\delta}^d - 1\}$.

For any two points $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ connected by a tree arc $(i,j) \in A_T$ the relation

$$\boldsymbol{v_j} = (\boldsymbol{v_i} + \boldsymbol{t_{\kappa_{i,j}}} + \boldsymbol{\delta_{i,j}}) \bmod \tilde{\boldsymbol{v}}, \qquad \forall (i,j) \in A_T, \tag{1}$$

must hold; i.e. $\boldsymbol{v_j}$ can be derived from $\boldsymbol{v_i}$ by adding the corresponding template and correction vectors. We use the modulo-calculation to avoid negative values and to be able to cross the domain-border within the arborescence. Our main objective now is to find a feasible solution with a smallest possible dictionary size m.

Details about encoding and decoding of the solution and the calculation and results regarding achieved compression ratios are given in the preceding work [1]. Here, we focus on an improved exact solution method, a new branch-and-cut-and-price algorithm.

A. The k-Minimum Label Spanning Arborescence Problem

Based on the corresponding section in [1], we summarize the problem formulation in the following. To be able to choose the root node of the arborescence by optimization we extend V to V^+ by adding an artificial root node 0. Further we extend A to A^+ by adding arcs $(0,i)$, $\forall i \in V$. We use the following variables for modeling the problem as an integer linear program (ILP):

- For each candidate template arc $\boldsymbol{t} \in T^c$, we define a variable $y_t \in \{0,1\}$, indicating whether or not the arc is part of the dictionary T.
- Further we use variables $x_{ij} \in \{0,1\}$, $\forall (i,j) \in A^+$, indicating which arcs belong to the tree.
- To express which nodes are covered by the tree, we introduce variables $z_i \in \{0,1\}$, $\forall i \in V$.

Let $A(t) \subset A$ denote the set of tree arcs a template arc $t \in T^c$ is able to represent when considering the allowed domains for the correction vectors, and let $T(a)$ be the set of template arcs that can be used to represent an arc $a \in A$, i.e. $T(a) = \{t \in T^c \mid a \in A(t)\}$. Hence, we can consider the template arcs $t \in T^c$ as the *labels* of their corresponding arcs $T(a)$, revealing the strong relation to the minimum label spanning tree problem.

We can now formulate the k-MLSA problem as follows:

$$\min \quad \sum_{t \in T^c} y_t \tag{2a}$$

$$\text{s.t.} \quad \sum_{t \in T(a)} y_t \geq x_a \qquad \forall a \in A \tag{2b}$$

$$\sum_{i \in V} z_i = k \tag{2c}$$

$$\sum_{a \in A} x_a = k - 1 \tag{2d}$$

$$\sum_{i \in V} x_{(0,i)} = 1 \tag{2e}$$

$$\sum_{(j,i) \in A+} x_{ji} = z_i \qquad \forall i \in V \tag{2f}$$

$$x_{ij} \leq z_i \qquad \forall (i,j) \in A \tag{2g}$$

$$x_{ij} + x_{ji} \leq z_i \qquad \forall (i,j) \in A \tag{2h}$$

$$\sum_{a \in C} x_a \leq |C| - 1 \quad \forall \text{ cycles } C \text{ in } G,\ |C| > 2 \tag{2i}$$

$$\sum_{a \in \delta^-(S)} x_a \geq z_i \qquad \begin{array}{l} \forall i \in V,\ \forall S \subseteq V, \\ i \in S,\ 0 \notin S \end{array} \tag{2j}$$

Inequalities (2b) ensure that for each used tree arc $a \in A$ at least one valid template arc t is selected. Equalities (2c) and (2d) enforce the required number of nodes and arcs to be selected. Equation (2e) requires exactly one arc from the artificial root to one of the tree nodes, which will be the actual root node of the outgoing arborescence.

Equations (2f) state that selected nodes must have in-degree one. Inequalities (2g) ensure, that an arc may only be selected if its source node is selected as well. Inequalities (2h) forbid cycles of length two, and finally inequalities (2i) forbid all further cycles ($|C| > 2$).

In order to strengthen the ILP we can additionally add (directed) connectivity-constraints, given by inequalities (2j), where $\delta^-(S)$ represents the ingoing cut of node set S. These constraints ensure the existence of a path from the root 0 to any node $i \in V$ for which $z_i = 1$, i.e. which is selected for connection. In principle, equations (2j) render (2f), (2g), (2h) and (2i) redundant [7], but using them jointly turned out to be sometimes beneficial in practice.

B. Candidate Template Arcs

The set of candidate template arcs T^c used in ILP (2) is, however, not explicitly given within the input data. The requirements to T^c are that it has to be sufficiently large to permit an overall optimal solution with regard to the pre-specified correction vector domain $\tilde{\boldsymbol{\delta}}$. Let $B = \{\boldsymbol{v_{ij}} = (\boldsymbol{v_j} - \boldsymbol{v_i}) \bmod \tilde{\boldsymbol{v}} \mid (i,j) \in A\} = \{\boldsymbol{b_1}, \dots, \boldsymbol{b_{|B|}}\}$ be the set of different vectors we eventually have to represent. Let further $D(\boldsymbol{t}) \subseteq \mathbb{D}$ be the subspace of all vectors a particular template arc $\boldsymbol{t} \in \mathbb{D}$ is able to represent when considering the restricted domain $\mathbb{D}'$ for the correction vectors, i.e. $D(\boldsymbol{t}) = \{t^1, \dots, (t^1 + \tilde{\delta}^1 - 1) \bmod \tilde{v}^1\} \times \dots \times \{t^d, \dots, (t^d + \tilde{\delta}^d - 1) \bmod \tilde{v}^d\}$. The subset of vectors from B that a particular template arc $\boldsymbol{t}$ is able to represent is denoted by $B(\boldsymbol{t}) = \{\boldsymbol{b} \in B \mid \boldsymbol{b} \in D(\boldsymbol{t})\}$. The set of candidate template arcs T^c must have the property that all possible elements $\boldsymbol{t}$ with mutually different and non-dominated $B(\boldsymbol{t})$ should be included. Within this context a template-arc $\boldsymbol{t'}$ is said do be dominated by $\boldsymbol{t''}$ iff $B(\boldsymbol{t'}) \subset B(\boldsymbol{t''})$.

In a previous branch-and-cut approach [1] the set of candidate template arcs T^c has been computed in a relatively time-consuming preprocessing step. Within the approach presented in this work, this preprocessing is not necessary anymore.

Suitable candidate template arcs are on demand derived in the pricing-step, described in Section III.

C. Cutting-plane Separation

The description of the cutting-plane separation again follows [1]. As there are exponentially many cycle elimination and connectivity inequalities (2i) and (2j), directly solving the ILP would be only feasible for very small problem instances. Instead, we apply branch-and-cut [8], i.e. we just start with the constraints (2b) to (2h) and add violated cycle elimination constraints and connectivity constraints only on demand during the optimization process.

Cycle elimination cuts (2i) can be easily separated by shortest path computations with Dijkstra's algorithm. Hereby we use $(1 - x_{ij}^{\mathrm{LP}})$ as the arc weights with x_{ij}^{LP} denoting the current value of the LP-relaxation for (i, j) in the current node of the branch-and-bound tree. We obtain cycles by iteratively considering each edge $(i, j) \in A$ and searching for the shortest path from j to i. If the value of a shortest path plus $(1 - x_{ij}^{\mathrm{LP}})$ is less than 1, we have found a cycle for which inequality (2i) is violated. We add this inequality to the LP and resolve it. In each node of the branch-and-bound tree we perform these cutting plane separations until no further cuts can be found.

The directed connection inequalities (2j) strengthen our formulation. Compared to the cycle elimination cuts they lead to better theoretical bounds, i.e. a tighter characterization of the spanning-tree polyhedron [7], but their separation usually is computationally more expensive. We separate them by computing the maximum flow (and therefore minimum $(0, i)$-cut) from the root node to each of the nodes with $z_i > 0$ as target node. If the value of this cut is less than z_i^{LP}, we have found an inequality that is violated by the current LP-solution. Our separation procedure utilizes Cherkassky and Goldberg's implementation of the push-relabel method for the maximum flow problem [9] to perform the required minimum cut computations.

The branch-and-cut algorithm has been implemented using C++ with CPLEX in version 11.2 [10].

III. Branch-and-Cut-and-Price

The former branch-and-cut algorithm presented in [1] has two major shortcomings. First, the preprocessing method is quite time-consuming, and second, the large amount of template-arc-variables yields large LPs to be solved within every node of the branch-and-bound tree. These observations support the idea to develop a branch-and-cut-and-price (BCP) approach, where after starting with a small, very restricted set of template arcs, further template arcs are dynamically added on demand. This approach has for the first time been investigated in the theses [4]–[6].

Following the idea of the valid inequalities proposed in [11], we introduce inequalities

$$\sum_{t \in T(\Gamma^-(v_i))} y_t \geq z_i - x_{ri}, \quad \forall i \in V, \tag{3}$$

to provide (besides inequalities (2b)) further information for the pricing step, but also to further strengthen the LP. Inequalities (3) state that for each selected node except the artificial root node, the sum over the template-arc variables associated to the nodes ingoing arcs, must be greater or equal than one.

A. Pricing Problem

Let π_a denote the dual variables corresponding to inequalities (2b), and μ_j denote the dual variables corresponding to inequalities (3). The reduced costs for each template arc $\boldsymbol{t}$ are then given by

$$\bar{c}_t = 1 - \left(\sum_{a \in A(t)} \pi_a + \sum_{j \in \{v | (u,v) \in A(t)\}} \mu_j \right). \tag{4}$$

Any template arc with negative reduced costs $\bar{c}_t$ may potentially improve the current objective function value; if there are no such template arcs, the solution cannot be further improved. We define the pricing problem as finding the template arc with maximal negative reduced costs.

Definition 1 (Pricing Problem):

$$\boldsymbol{t}^* = \operatorname{argmin}_{t \in T} \left\{ 1 - \left(\sum_{t \in A(t)} \pi_a + \sum_{j \in \{v | (,v) \in A(t)\}} \mu_j \right) \right\} \tag{5}$$

B. Solving the Pricing Problem

A nice geometrical interpretation for the pricing problem arises, when considering the two-dimensional case. Each tree arc corresponds to a point in $\mathbb{D}$ according to its associated geometric information. Furthermore, each point in $\mathbb{D}$ in the same way corresponds to a potential template arc. Hence, we will use the terms tree/template arc and their corresponding points interchangeably within this section. All template arcs potentially representing an arbitrary tree arc $\boldsymbol{b}$ must have their endpoint in the rectangle $D(\boldsymbol{b} - \tilde{\boldsymbol{\delta}} + \mathfrak{e})$ with $\boldsymbol{b}$ corresponding to its upper right corner, and $\mathfrak{e}$ denoting the d-dimensional vector with all components being one. Let $T'(\boldsymbol{b})$ denote this area whose points correspond to the potential template arcs able to represent $\boldsymbol{b}$. To each $T'(\boldsymbol{b})$ we now associate the value

$$\zeta_b = \sum_{i \in \{a | a \in A \wedge a = b\}} \pi_i + \sum_{j \in \{v | (u,v) \in A \wedge (u,v) = b\}} \mu_j. \tag{6}$$

The first term on the right hand side in (6) corresponds to the sum of all dual values associated to the constraints for the tree arcs corresponding to $\boldsymbol{b}$, given by inequalities (2b). The second term results from the dual values of all nodes according to constraints (3) which are incident to a tree arc corresponding to $\boldsymbol{b}$. We can now imagine these rectangles $T'(\boldsymbol{b})$ as transparently shaded with a color of intensity ζ_b, where higher values corresponding to darker shades. See Fig. 1 for an example of two elements $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ and their corresponding regions $T'(\boldsymbol{b_1})$ and $T'(\boldsymbol{b_2})$ being drawn in the domain.

Let us now consider the situation of all $\boldsymbol{b} \in B$ and their corresponding $T'(\boldsymbol{b})$, shaded accordingly with ζ_b, being drawn in the area corresponding to the two-dimensional domain $\mathbb{D}$. Due to the transparency of the rectangles, regions of

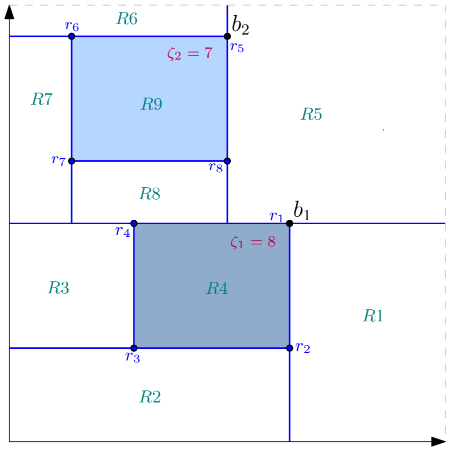

Fig. 1. Example of two elements $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ and corresponding regions $T'(\boldsymbol{b_1})$ and $T'(\boldsymbol{b_2})$ drawn in the domain $\mathbb{D}$. (Image with minor modifications taken from [4])

overlapping rectangles will obtain darker colors. Formally we define for each uniform region R a corresponding value

$$\zeta_R = \sum_{b \in A(t),\ t \in R} \zeta_b \tag{7}$$

for some arbitrary $\boldsymbol{t}$ being located in region R. Figure 2 shows an example with two overlapping elements $\boldsymbol{b_i}$. We can now see that the pricing problem given by Definition 1 exactly corresponds to finding the darkest such area. This analogy remains valid even in the higher dimensional case if we use regions of corresponding dimensionality instead of areas with dimensionality two. The correspondence of the presented illustration to the pricing problem becomes evident by considering the correspondence of equation (6) to the two sums in (5). The only difference is that (6) is formulated in terms of unique points $\boldsymbol{b}$ and (5) in terms of template arcs $\boldsymbol{t}$, which we actually want to determine.

Based on this observation, we now outline an algorithm to solve the pricing problem. This algorithm was primary subject of the diploma thesis [4]; for details according to the implementation of the algorithm, the reader is referred to this work.

The underlying data structure is a k-d tree [12] which is used to partition the domain into the corresponding regions resulting from $T'(\boldsymbol{b})$, for all $\boldsymbol{b} \in B$ and resulting overlapping regions. Here k denotes the number of dimensions to be used within the tree, and should not be mixed up with the number of nodes to be connected to the arborescence. However, as the term k-d tree is commonly used for this data structure, we refrain from referring to it as d-d tree. For convenience, we briefly review the principles of k-d trees. Their primary application is to store multidimensional data points and allow efficient range and nearest neighbor searches. The tree defines a hierarchical partitioning of the underlying domain. Each node of the binary tree defines a division of the subspace in which it is located into exactly two further subspaces. Within each level l coordinate $l \bmod k$ is used to define this subdivision. At the root node the whole domain is subdivided according to some coordinate of the first dimension. Each child node then defines a subdivision according to a coordinate of the second dimension, and so forth. For our purpose we define each node to have either two children, or to be a leaf node. In our case, a leaf node may either correspond to a region that cannot contain a template arc, or otherwise, a region that contains all possible template arcs representing a unique subset of elements $\boldsymbol{b_i} \in \mathbb{B}$.

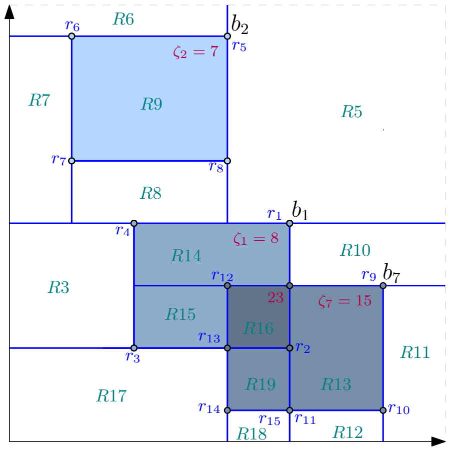

Fig. 2. This illustration shows the situation after the insertion of a further element ($\boldsymbol{b_7}$) into the segmentation tree shown in Figures 1 and 3. (Image with minor modifications taken from [4])

Figures 1 and 2 show examples of two and three elements being drawn in the domain, respectively. These figures illustrate how the domain is segmented into subregions according to $T'(\boldsymbol{b_1})$ and $T'(\boldsymbol{b_2})$ (and $T'(\boldsymbol{b_7})$ in Figure 2). Corresponding k-d trees, which we will from now on call *segmentation trees*, are depicted in Figures 3 and 4. As each node of the tree subdivides its subspace into two subspaces, it defines a *hyperplane*, which we will also call *splitting-hyperplane*. In the two-dimensional examples of Figures 1 and 2 these hyperplanes correspond to lines. In the example of Figure 1 the first split is performed according to the second coordinate at point $\boldsymbol{r_1} = \boldsymbol{b_1}$. Nodes $\boldsymbol{r_i}$ denote the coordinates corresponding to each node i of the segmentation tree. The area $T'(\boldsymbol{b_1})$ is finally defined by nodes $\boldsymbol{r_2}$, $\boldsymbol{r_3}$ and $\boldsymbol{r_4}$, area $T'(\boldsymbol{b_2})$ by nodes $\boldsymbol{r_5}$, $\boldsymbol{r_6}$, $\boldsymbol{r_7}$ and $\boldsymbol{r_8}$. In the figure all points $\boldsymbol{r_i}$ correspond to the corner points of areas $T'(\boldsymbol{b})$ for all elements $\boldsymbol{b}$ in the tree, which is, however, an arbitrary decision for a better illustration. In fact, only one coordinate is required to define a hyperplane being orthogonal to the basis vector of the considered dimension, which is always the case in the segmentation tree. Besides the intermediate "splitting" nodes, the tree in Figure 3 also contains the leaf nodes, with corresponding regions depicted in Figure 1. The second example, given by Figures 2 and 4 shows the resulting tree after the insertion of element $\boldsymbol{b_7}$. Again, splitting nodes and leafs (corresponding to regions) are contained in the visualization of the tree, as well as in

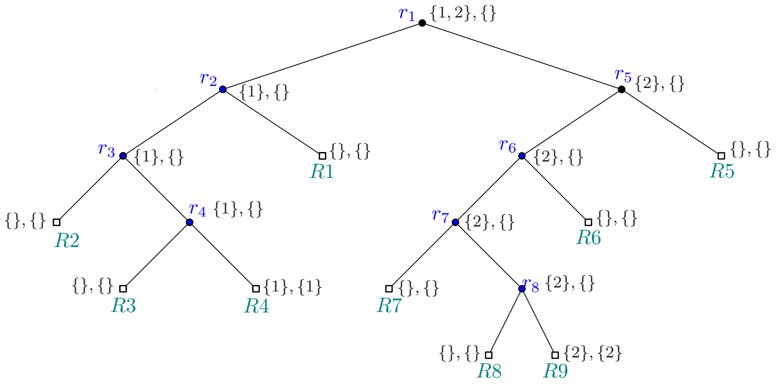

Fig. 3. Segmentation tree corresponding to the example shown in Figure 1. (Image credits: [4])

the corresponding illustration of the fragmented domain. To build up the whole tree, regions $T'(\boldsymbol{b})$ for all $\boldsymbol{b} \in B$ are iteratively inserted. For each such $T'(\boldsymbol{b})$ we need to find the correct position for inserting it into the tree. This is done by checking at each tree node $\boldsymbol{r}$ if $T'(\boldsymbol{b})$ is entirely located in one of the subspaces defined by $\boldsymbol{r}$. If a region is entirely contained in a region defined by a current leaf of the tree, this leaf is replaced with an according subtree corresponding to the splitting hyperplanes required to properly define $T'(\boldsymbol{b})$. However, $T'(\boldsymbol{b})$ is part of both subspaces defined by the current node $\boldsymbol{r}$, we need to split $T'(\boldsymbol{b})$ accordingly, and insert the resulting subregions into both branches of $\boldsymbol{r}$. Having now described, how the segmentation tree is created, we focus on how the tree can be used to efficiently search for the best template arc.

At this point we assume that the whole segmentation tree has been created in advance. As we will see later, this is not a real requirement. Our goal is to find the region R with maximum ζ_R, which is the solution to the pricing problem. As the pricing problem needs to be solved many times, the search has to be efficient. In particular we want to avoid to assign ζ_B to all leafs (corresponding to regions) B in the tree according to the values ζ_b derived by the dual values. Therefore the search is based on upper and lower bounds used to prune branches at an early stage. Let $R(\boldsymbol{r})$ denote the subspace corresponding to node $\boldsymbol{r}$ of the segmentation tree. Upper and lower bounds for each node $\boldsymbol{r}$ of the tree can be derived based on the following definitions.

Definition 2 (Upper Bound Set): The upper bound set is given by all elements $\boldsymbol{b} \in B$ which can be represented by *some* potential template arc in the subspace corresponding to tree node $\boldsymbol{r}$.

$$UB(\boldsymbol{r}) = \{\boldsymbol{b} \in B \mid \exists \boldsymbol{t} \in R(\boldsymbol{r}) \wedge \boldsymbol{b} \in B(\boldsymbol{t})\}$$

Definition 3 (Lower Bound Set): The lower bound set is given by all elements $\boldsymbol{b} \in B$ which can be represented by *all* potential template arcs in the subspace corresponding to tree node $\boldsymbol{r}$.

$$LB(\boldsymbol{r}) = \{\boldsymbol{b} \in B \mid \forall \boldsymbol{t} \in R(\boldsymbol{r}) \wedge \boldsymbol{b} \in B(\boldsymbol{t})\}$$

These bound sets are stored for each node of the search tree. In Figures 1 and 2 these sets are denoted in braces at each node.

Based on these sets we can immediately derive numeric bounds, based on the dual values.

Definition 4 (Upper Bound):

$$ub(\boldsymbol{r}) = \sum_{\boldsymbol{b} \in UB(\boldsymbol{r})} \zeta_{\boldsymbol{b}}$$

Definition 5 (Lower Bound):

$$lb(\boldsymbol{r}) = \max_{\boldsymbol{b} \in UB(\boldsymbol{r})} \zeta_{\boldsymbol{b}}$$

The search process is performed based on these upper and lower bounds. Starting at the root node, the set B is divided into two not necessarily disjoint sets. These sets $UB(\boldsymbol{r})$ correspond to the nodes which are representable by some template arc of the subspaces introduced by the splitting-hyperplane defined by the current tree node $\boldsymbol{r}$. With $ub(\boldsymbol{r})$ we directly obtain a numeric value being the upper bound for this particular branch. A lower bound is given by $lb(\boldsymbol{r})$, i.e. the element with maximal $\zeta_{\boldsymbol{b}}$ in this branch. For each node we check if $UB(\boldsymbol{r}) = LB(\boldsymbol{r})$ which implies that we have found a leaf node. A global lower bound lb^* is used to prune the search tree, as we do not have to follow branches with $ub(\boldsymbol{r}) < lb^*$. Initialization of the global lower bound can be performed with $lb^* = \max_{\boldsymbol{b} \in B} \zeta_{\boldsymbol{b}}$. The search strategy to be used is *best first search* based on the upper bounds $ub(\boldsymbol{r})$.

Within the description of the algorithm, we have omitted many implementation issues. One important aspect to be considered is the fact that regions may cross the domain border. This needs to be checked in advance, and corresponding subregions must be inserted in this case. Furthermore a lot of design issues are involved in order to implement the bounding procedure efficiently. Also the reconstruction of the coordinate values of the corner points of each region requires to take care of some special cases. For a detailed presentation and analysis of this issues we refer to [4].

A further substantial improvement of the overall process can be achieved if the entire tree is not completely built in advance, but rather in a dynamic on demand way during the

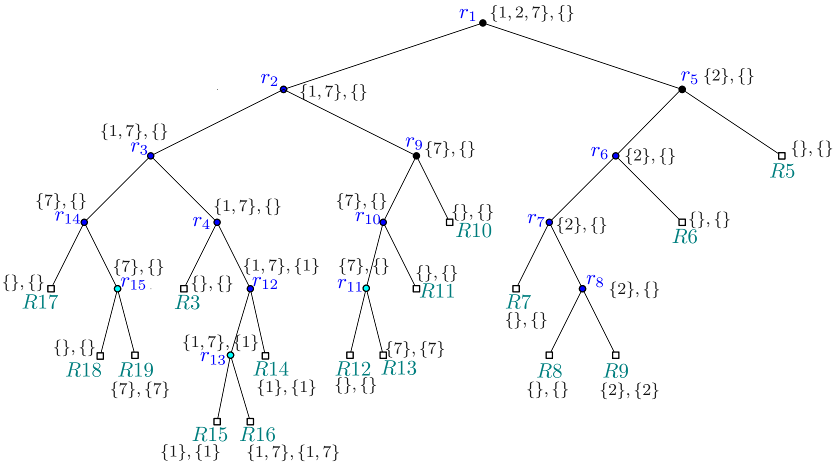

Fig. 4. Segmentation tree corresponding to the example shown in Figure 2. (Image credits: [4])

search process. Each time the search is according to the bounds directed toward a certain branch of the tree, we check if this branch has already been created. If this is not the case, it is expanded as needed. Hence construction and traversing the tree is performed in an intertwined way. This has not only the advantage of the initial construction step to be omitted, but will also result in smaller trees to operate with. As certain regions of the domain will not contain any useful template arcs, corresponding branches are unlikely to be created during the whole BCP solution process, saving space and time.

Corresponding pseudocodes are omitted within this presentation, as they would require a more detailed formal description and notation. In the following section we show how this algorithmic framework for solving the pricing problem can be used within a branch-and-cut-and-price approach.

C. Branch-and-Cut-and-Price Algorithm

The first step of the entire branch-and-cut-and-price (BCP) is to determine a feasible starting solution. Any connected subgraph of k nodes is sufficient for this purpose. Hence, we determine a starting solution by connecting arbitrary k nodes by a star-shaped spanning tree, assign big values to the dual variables corresponding to this set of arcs, and use the pricing algorithm to determine a feasible starting solution.

The restricted master problem (RMP) is defined according to the ILP from Section II-A, however the entire set T^c is replaced by T^p denoting the set of template arc variables that have already been priced in. Within each node of the branch-and-bound-tree directed connection cuts and cycle elimination cuts are separated to obtain a feasible LP-relaxation. Afterwards new template arc variables are priced in as long as such variables with negative reduced costs according to Equation (5) can be found and no further cutting-planes can be added. It turned out to be advantageous to add all variables with negative reduced costs within each pricing iteration.

IV. Results

In this section we present the results of our computational experiments with the outlined branch-and-cut-and-price algorithm. For this purpose two different data sets have been used. The first set of 20 instances was provided by the Fraunhofer Institute Berlin and is in the following referred to as *Fraunhofer Templates*. Furthermore 14 randomly selected instances from the U.S. National Institute of Standards and Technology [13] have been used.

All test runs have been performed on an Intel Core 2 Quad running at 2.83 GHz with 8 GB RAM and Ubuntu 11.04. The branch-and-cut-and-price algorithm has been implemented in C++ within the SCIP framework [14]. and CPLEX in version 11.2 [10], which is also used for the comparison to the branch-and-cut (BC) algorithm from [1]

For each run a time-limit of two hours has been imposed. Table I shows average solution values for various parameter settings (k, $\tilde{\boldsymbol{\delta}}$) and groups of instances. These averages are taken over all instances that have been solved within the time limit (indicated in the last column). The Fraunhofer instances with $|V| < 30$ are not included in the corresponding groups with $k = 30$. Column "pit" reports the average numbers of pricing iterations, column "pvar" the average numbers of priced in variables, and column "bbn" the average branch and bound nodes. Column "cuts" reports the numbers of applied cuts, which consist of directed connection cuts ("DCC") and cycle elimination cuts ("CEC").

Table II shows the comparison of the new BCP algorithm to the branch-and-cut algorithm presented in [1]. The average running times of the BC algorithm do not include the preprocessing step. The reported average running times and numbers of branch-and-bound nodes are not directly comparable, as not always the same number of instances has been solved by both algorithms. The BCP method is, however, clearly superior to the BC algorithm w.r.t. the number of solved instances, and also yields to significantly lower average running times and numbers of branch-and-bound nodes.

TABLE I
RESULTS OF THE BRANCH-AND-CUT-AND-PRICE ALGORITHM. AVERAGE VALUES FOR ALL SOLVED INSTANCES IN THE PARTICULAR GROUP ARE REPORTED.

Instances	Parameters	avg t[s]	pit	pvar	bbn	cuts	DCC	CEC	inst.solved
Fraunhofer	$\tilde{\delta}^\top = (10,10,10), k = 20$	7.7	134	86	39	113	100	13	20/20
Fraunhofer	$\tilde{\delta}^\top = (10,10,10), k = 30$	4.7	90	65	18	188	167	21	18/18
NIST	$\tilde{\delta}^\top = (10,10,10), k = 40$	2945.0	20006	311	19636	1150	1061	89	5/15
NIST	$\tilde{\delta}^\top = (10,10,10), k = 80$	886.6	11950	174	11616	8032	7625	407	12/15
NIST	$\tilde{\delta}^\top = (10,10,10), k = \|V\|$	237.1	1665	162	1464	3702	3400	302	11/14
Fraunhofer	$\tilde{\delta}^\top = (20,20,20), k = 20$	23.0	304	174	120	164	134	30	20/20
Fraunhofer	$\tilde{\delta}^\top = (20,20,20), k = 30$	14.4	253	162	79	280	224	56	18/18
NIST	$\tilde{\delta}^\top = (20,20,20), k = 40$	890.0	1067	632	399	1192	972	220	6/15
NIST	$\tilde{\delta}^\top = (20,20,20), k = 80$	987.8	691	412	257	1532	1356	177	14/15
NIST	$\tilde{\delta}^\top = (20,20,20), k = \|V\|$	232.7	496	353	125	1653	1493	159	14/14
Fraunhofer	$\tilde{\delta}^\top = (30,30,30), k = 20$	132.3	1379	594	762	410	305	105	20/20
Fraunhofer	$\tilde{\delta}^\top = (30,30,30), k = 30$	28.0	537	311	207	552	447	105	18/18
NIST	$\tilde{\delta}^\top = (30,30,30), k = 40$	2970.0	2647	960	1591	3724	2995	729	6/15
NIST	$\tilde{\delta}^\top = (30,30,30), k = 80$	2219.3	1873	767	988	8688	7608	1079	12/15
NIST	$\tilde{\delta}^\top = (30,30,30), k = \|V\|$	2318.3	3032	986	1997	4580	4124	457	11/14
Fraunhofer	$\tilde{\delta}^\top = (40,40,40), k = 20$	163.3	2069	1119	936	212	167	44	20/20
Fraunhofer	$\tilde{\delta}^\top = (40,40,40), k = 30$	148.3	1195	709	474	273	220	53	18/18
NIST	$\tilde{\delta}^\top = (40,40,40), k = 40$	3139.8	3556	1124	2223	9304	7193	2111	4/15
NIST	$\tilde{\delta}^\top = (40,40,40), k = 80$	1808.5	2598	982	1492	9535	8368	1166	6/15
NIST	$\tilde{\delta}^\top = (40,40,40), k = \|V\|$	2844.3	5258	1265	3919	5940	5154	786	5/14
Fraunhofer	$\tilde{\delta}^\top = (50,50,50), k = 20$	47.5	547	451	88	118	97	21	18/20
Fraunhofer	$\tilde{\delta}^\top = (50,50,50), k = 30$	83.6	886	645	230	285	223	62	18/18
NIST	$\tilde{\delta}^\top = (50,50,50), k = 40$	2996.8	3566	1477	1901	7411	5832	1579	2/15
NIST	$\tilde{\delta}^\top = (50,50,50), k = 80$	1313.5	3546	1089	2397	4589	3995	594	2/15
NIST	$\tilde{\delta}^\top = (50,50,50), k = \|V\|$	4190.9	7542	1747	5645	13019	11098	1921	1/14
Fraunhofer	$\tilde{\delta}^\top = (60,60,60), k = 20$	588.4	1869	1656	202	173	134	39	16/20
Fraunhofer	$\tilde{\delta}^\top = (60,60,60), k = 30$	65.5	791	574	199	472	358	114	18/18
NIST	$\tilde{\delta}^\top = (60,60,60), k = 40$	5177.2	5740	1951	3712	2725	2091	634	1/15
NIST	$\tilde{\delta}^\top = (60,60,60), k = 80$	2039.9	3928	1450	2469	659	583	76	3/15
NIST	$\tilde{\delta}^\top = (60,60,60), k = \|V\|$	4066.7	6836	2083	4709	3764	3292	472	4/14

The results clearly show that the new BCP approach is able to solve a significantly larger number of instances and also requires shorter running times on average for most classes of instances.

V. CONCLUSIONS

In this work we have presented a branch-and-cut-and-price framework to solve the problem of compressing a relatively small unordered set of multidimensional points with the application background of embedding fingerprint minutiae data into passport images by watermarking techniques as an additional security feature. Compared to a previously used exact branch-and-cut algorithm, a significant speedup of solving the underlying combinatorial optimization problem (k-MLSA problem) could be achieved. Furthermore the preprocessing step beeing necessary for the preceding approach needs not to be performed anymore. As a result more instances can be solved to proven optimality within the considered time limit by the new method.

Although the overall compression ratios achieved by our particular model are rather limited, they are clearly superior to other popular compression mechanisms, which cannot perform any compression on the considered data at all. Further improvements regarding compression ratios can possibly be achieved by devising refined models and algorithms. However, in consideration of being a new approach to data compression by combinatorial optimization techniques, as well as being a novel approach of directly exploiting the property that the order of the underlying data needs not to be preserved, our approach can be regarded a successful proof-of-concept to be able to compress weakly structured data sets and appears to be a promising origin for further research in the field of combinatorial optimization based compression methods.

TABLE II
COMPARISON OF THE RESULTS ACHIEVED BY BRANCH-AND-CUT-AND-PRICE AND THE BRANCH-AND-CUT.

Instances	Parameters	branch-and-cut			branch-and-cut-and-price		
		avg t[s]	bbn	inst.solved	avg t[s]	bbn	inst.solved
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (10, 10, 10), k = 20$	3.8	27	20/20	7.7	39	20/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (10, 10, 10), k = 30$	4.4	253	18/18	4.7	18	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (10, 10, 10), k = 40$	2889.4	1574	1/15	2945.0	19636	5/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (10, 10, 10), k = 80$	1416.4	6371	4/15	886.6	11616	12/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (10, 10, 10), k = \|V\|$	498.7	623	3/14	237.1	1464	11/14
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (20, 20, 20), k = 20$	28.0	579	20/20	23.0	120	20/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (20, 20, 20), k = 30$	11.3	191	18/18	14.4	79	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (20, 20, 20), k = 40$	2220.7	1691	8/15	890.0	399	6/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (20, 20, 20), k = 80$	537.2	157	13/15	987.8	257	14/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (20, 20, 20), k = \|V\|$	322.4	69	13/14	232.7	125	14/14
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (30, 30, 30), k = 20$	76.6	1513	20/20	132.3	762	20/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (30, 30, 30), k = 30$	98.7	2657	18/18	28.0	207	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (30, 30, 30), k = 40$	2922.5	3860	4/15	2970.0	1591	6/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (30, 30, 30), k = 80$	1291.0	880	9/15	2219.3	988	12/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (30, 30, 30), k = \|V\|$	2281.8	2515	8/14	2318.3	1997	11/14
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (40, 40, 40), k = 20$	241.4	5471	20/20	163.3	936	20/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (40, 40, 40), k = 30$	256.4	3493	18/18	148.3	474	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (40, 40, 40), k = 40$	2712.3	4218	2/15	3139.8	2223	4/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (40, 40, 40), k = 80$	1071.6	1296	5/15	1808.5	1492	6/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (40, 40, 40), k = \|V\|$	2762.4	4232	3/14	2844.3	3919	5/14
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (50, 50, 50), k = 20$	292.4	2246	13/20	47.5	88	18/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (50, 50, 50), k = 30$	604.1	6606	9/18	83.6	230	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (50, 50, 50), k = 40$	3681.7	5030	1/15	2996.8	1901	2/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (50, 50, 50), k = 80$	1611.7	5991	2/15	1313.5	2397	2/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (50, 50, 50), k = \|V\|$	1711.7	3633	1/14	4190.9	5645	1/14
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (60, 60, 60), k = 20$	864.7	2425	12/20	588.5	202	16/20
Fraunhofer	$\tilde{\boldsymbol{\delta}}^{\top} = (60, 60, 60), k = 30$	710.1	5504	11/18	65.5	199	18/18
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (60, 60, 60), k = 40$	1854.6	5652	1/15	5177.2	3712	1/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (60, 60, 60), k = 80$	3073.4	4121	1/15	2039.9	2469	3/15
NIST	$\tilde{\boldsymbol{\delta}}^{\top} = (60, 60, 60), k = \|V\|$	3069.7	4544	1/14	4066.7	4709	4/14

REFERENCES

[1] A. M. Chwatal, G. R. Raidl, and K. Oberlechner, "Solving a k-node minimum label spanning arborescence problem to compress fingerprint templates," *Journal of Mathematical Modelling and Algorithms*, vol. 8, pp. 293–334, 2009.

[2] R.-S. Chang and S.-J. Leu, "The minimum labeling spanning trees," *Information Processing Letters*, vol. 63, no. 5, pp. 277–282, 1997.

[3] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel, "Obtaining optimal k-cardinality trees fast," *Journal of Experimental Algorithmics*, vol. 14, pp. 5.1–5.23, 2010.

[4] C. Thöni, "Compressing fingerprint templates by solving the k-node minimum label spanning arborescence problem by branch-and-price," Master's thesis, Vienna University of Technology, Vienna, Austria, 2010.

[5] K. Oberlechner, "Solving a k-node minimum label spanning arborescence problem with exact and heuristic methods," Master's thesis, Vienna University of Technology, Vienna, Austria, 2010.

[6] A. M. Chwatal, "On the minimum label spanning tree problem: Solution methods and applications," Ph.D. dissertation, Vienna University of Technology, 2010.

[7] T. Magnanti and L. Wolsey, "Optimal trees," *Handbook in Operations Research and Management Science*, vol. Network Models, pp. 503–615, 1995.

[8] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. Wiley-Interscience, November 1999. [Online]. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&amp;path=ASIN/0471359432

[9] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997. [Online]. Available: citeseer.ist.psu.edu/cherkassky94implementing.html

[10] ILOG Concert Technology, CPLEX, "ILOG," http://www.ilog.com, version 11.0.

[11] A. M. Chwatal and G. R. Raidl, "Solving the minimum label spanning tree problem by mathematical programming techniques," *Advances in Operations Research*, 2011, (in press).

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commununications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[13] Garris M. D. and McCabe R. M., "NIST special database 27: Fingerprint minutiae from latent and matching tenprint images." National Institute of Standards and Technology, Tech. Rep., 2000.

[14] SCIP – Solving Constraint Integer Programs, "ILOG," http://scip.zib.de/, version 1.2.